

IN THE
UNITED STATES PATENT AND TRADEMARK OFFICE

Inventor(s) Sanjeev BANERJIA et al.

Confirmation No.: 5215

Application No.: 09/755,389

Examiner: A. Fowlkes

Filing Date: Jan. 5, 2001

Group Art Unit: 2122

Title: A PARTITIONED CODE CACHE ORGANIZATION TO EXPLOIT PROGRAM LOCALITY

Mail Stop Appeal Brief-Patents
Commissioner For Patents
PO Box 1450
Alexandria, VA 22313-1450

TRANSMITTAL OF APPEAL BRIEF

Sir:

Transmitted herewith is the Appeal Brief in this application with respect to the Notice of Appeal filed on 02/17/2005.

The fee for filing this Appeal Brief is (37 CFR 1.17(c)) \$500.00.

(complete (a) or (b) as applicable)

The proceedings herein are for a patent application and the provisions of 37 CFR 1.136(a) apply.

() (a) Applicant petitions for an extension of time under 37 CFR 1.136 (fees: 37 CFR 1.17(a)-(d) for the total number of months checked below:

() one month	\$120.00
() two months	\$450.00
() three months	\$1020.00
() four months	\$1590.00

() The extension fee has already been filled in this application.

(X) (b) Applicant believes that no extension of time is required. However, this conditional petition is being made to provide for the possibility that applicant has inadvertently overlooked the need for a petition and fee for extension of time.

Please charge to Deposit Account **08-2025** the sum of \$500.00. At any time during the pendency of this application, please charge any fees required or credit any over payment to Deposit Account 08-2025 pursuant to 37 CFR 1.25. Additionally please charge any fees to Deposit Account 08-2025 under 37 CFR 1.16 through 1.21 inclusive, and any other sections in Title 37 of the Code of Federal Regulations that may regulate fees. A duplicate copy of this sheet is enclosed.

() I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail in an envelope addressed to:
Commissioner for Patents, Alexandria, VA
22313-1450. Date of Deposit: _____

OR

() I hereby certify that this paper is being transmitted to the Patent and Trademark Office facsimile number _____ on _____

Number of pages:

Typed Name:

Signature: _____

Respectfully submitted,

Sanjeev BANERJIA et al.

By _____

William T. Ellis

Attorney/Agent for Applicant(s)

Reg. No. 26,874

Date: 2/18/2005 (Mon)



Attorney Docket 10990960-1

**IN THE UNITED STATES PATENT AND TRADEMARK OFFICE
BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES**

Applicant: Sanjeev BANERJIA et al.
Title: A PARTITIONED CODE CACHE ORGANIZATION TO EXPLOIT
PROGRAM LOCALITY
Appl. No.: 09/755,389
Filing Date: January 5, 2001
Examiner: A. Fowlkes
Art Unit: 2122

APPEAL BRIEF UNDER 37 C.F.R. § 41.37

Commissioner for Patents
Washington, D.C. 20231

Sir:

The following is the Appellant Appeal Brief under the provisions of 37 C.F.R.41.37.

1. Real Party in Interest

The real party in interest is the assignee of record, Hewlett-Packard Development Company L.P., a Texas Limited Liability Partnership.

2. Related Appeals and Interferences

There are no related appeals or interferences that will directly affect, be directly affected by or have a bearing on the present appeal, that are known to appellant, the assignee, or the appellant's patent representative.

3. Status of Claims

The present appeal is directed to claims 1-23 which are the claims under consideration. A copy of the pending claims 1-23 are attached herein in the Claims Appendix (Section 10).

Claims 1-10, 12-20, 22, and 23 are finally rejected under 35 U.S.C. § 102(e) as being anticipated by U.S. patent no. 6,330,556 ("Chilimbi"). Claims 11 and 21 are rejected under

04/19/2005 MGBREM1 00000029 082025 09755389

01 FC:1402 500.00 DA
002.1359805.1

35 U.S.C. § 103(a) as being unpatentable over Chilimbi, in view of U.S. patent no. 5,675,790 ("Walls").

4. Status of Amendments

Claims 1-23 were initially pending in the application filed on January 5, 2001. Claim 23 was amended in an Amendment and Reply Under 37 C.F.R. § 1.111 filed March 18, 2004 in reply to a first Office Action on the merits mailed on December 22, 2003.

Claims 1, 2, 5-7, 9-14, and 17-23 were amended in an Amendment and Reply Under 37 C.F.R. § 1.111 filed August 10, 2004 in reply to a second Office Action on the merits mailed on May 11, 2004.

A Reply Under 37 C.F.R. § 1.116 was filed on January 5, 2005, in reply to a final Office Action mailed on November 17, 2004, which rejected all of the claims. An Advisory Action was mailed on January 26, 2005, which stated that the application was not in condition for allowance after consideration of the reply.

5. Summary of the Invention

Independent claims 1, 12, and 23 are directed to a method, system, and computer program product that operates a code cache in a dynamic instruction translator. The operation of the code cache includes storing a plurality of instruction translations in a cold partition of a cache memory. A determination is made whether an instruction translation that has been stored in the cold partition is hot. If so, the instruction translation that is determined to be hot is moved to a hot partition in the cache memory.

Figure 1 illustrates an interpreter 11 that receives and interprets an instruction input stream 16. A trace selector 12 identifies instruction traces that are to be stored in the code cache 13. See page 5, lines 10-28.

As described in page 6, lines 10-22, the code cache is partitioned into disjoint areas of memory and translations are stored in the specific partitions of the code cache based on the frequency of execution of the translations. In one embodiment, the code cache is partitioned into two partitions, a cold partition and a hot partition. See page 6, lines 23-25.

The process of determining whether an instruction translation is hot and moving the hot instruction translation into the hot partition of the cache is discussed with respect to Figure 2 and its description in the specification. Specifically, all new translations are initially

stored in the cold partition of the cache. See steps 100 and 102 in Fig. 2 and page 7, lines 9-11. After the translation is executed in step 104 the counter association with the translation is incremented in step 108. See page 7, lines 25-27.

If the incremented counter value exceeds a hot threshold, the translation is moved from the cold cache partition to the hot cache partition as shown in steps 110 and 114. See page 8, lines 1-15.

6. Issues

The issue on appeal is whether the examiner erred in rejecting claims 1-10, 12-20, 22, and 23 under 35 U.S.C. § 102(e) as being anticipated by Chilimbi and in rejecting claims 11 and 21 under 35 U.S.C. § 103(a) as being unpatentable over Chilimbi in view of Walls.

7. Grouping of Claims

Claims 1-23 do not stand or fall together. Rather, the following grouping of claims are provided, whereby separate arguments are provided below for each of these groups:

Group 1 – Claims 1-9, 12-19, and 22-23

Group 2 – Claims 10-11 and 20-21

8. Argument

Group 1

It is respectfully submitted that the final rejection of claims 1-9, 12-19, and 22-23 as being anticipated by Chilimbi is erroneous for at least the following reasons.

(A) Each of the independent claims 1, 13, and 23 recite a method (or system/software) that operates a code cache in a dynamic instruction translator which (1) stores a plurality of *instruction* translations in a cold partition of the *code* cache, (2) determines whether the *instruction* translation stored in the cold partition is hot; and (3) moves the *instruction* translation to a hot partition when an instruction translation has been determined to be hot. None of the applied references relate to a code cache and, therefore, necessarily do not teach or suggest any of the claimed features related to the storage and movement of instruction translations between hot and cold partitions of a code cache.

Specifically, Chilimbi relates to partitioning of *data* structures into heavily referenced and less referenced portions with the heavily referenced *data* being kept in a hot object or partition. See col. 2, lines 36-43 of Chilimbi. This storing and partitioning of data in the cache is made clear through out the Chilimbi specification. For example, as shown in Figs. 2 and 3, Chilimbi discloses the data elements a, b, and c (of data structure A) and data elements x and y (of data structure B) being related in a field affinity graph of Fig. 3 so that data that have a temporal affinity can be co-located in a cache. See, for example, cols. 6-9 of Chilimbi that describe this process. Therefore, nowhere does Chilimbi teach or suggest management for a *code* cache in which instruction translations are stored and moved between cold and hot partitions as recited in the pending independent claims. Therefore, Chilimbi does not anticipate the pending independent claims.

(B) The assertion in the final office action dated November 17, 2004, and in the Advisory Action dated January 26, 2005 that Chilimbi discloses moving code within a code cache from a cold partition to a hot partition is erroneous for at least the following reasons.

First, the office action misquotes (by quoting the first part of a sentence but omitting the relevant second part of the same sentence) Chilimbi at col. 2, lines 14-17 for the proposition that Chilimbi discloses that classes include both data and code and that Chilimbi therefore allegedly discloses partitioning code as conjectured in the office action.

Col. 2, lines 14-18 of Chilimbi states “Classes define containers of data or information and code which operates on the data in response to method calls from other users or classes.” (emphasis added). Therefore, Chilimbi makes a very clear distinction between the data and code that may be contained in a class and specifically and repeatedly teaches that the partitioning is performed on data to improve access to data. For example, with regard to problem that is sought to solved, Chilimbi states “[i]f there is insufficient space available for desired *data*, time is spend in obtaining the desired *data* from the slower storage and then populating the cache lines so that the desired *data* is more quickly available to the processor.” (emphasis added). See col. 2, lines 26-30 of Chilimbi. The summary of invention of Chilimbi states that “[d]ata structures are partitioned into heavily referenced and less references portions.” See col. 2, lines 35-37. In the field reordering example shown in Figures 2 and 3 of Chilimbi all the fields reordered are data fields. See cols. 6 and 7 of Chilimbi. In fact, Chilimbi later clarifies in col. 11, that the term “field” refers to class

instance *variables* which are of course “data” as would be recognized by one skilled in the computing sciences or programming art. See col. 11, lines 12-14 of Chilimbi. With respect to the “caches,” Chilimbi also clarifies that “[c]aches have finite associativity, which means that only a limited number of concurrently accessed *data* elements can map to the same cache block without incurring conflict misses.” (emphasis added). See col. 13, lines 26-29 of Chilimbi. Based on this extensive evidence, it is very clear that Chilimbi appreciated the difference between data and code in a class (or object) but its disclosure is only directed at partitioning data in data caches and in no way teaches or suggests the features of the claimed invention.

Second, the office action’s assertion that Chilimbi teaching that it applies to Java programs somehow teaches code caches is also erroneous because the Chilimbi only teaches the use of Java programs as one example of an object oriented environment in which the data in the Java classes can be partitioned consistent with the Chilimbi invention. For example, in col. 6, lines 9-11, Chilimbi discloses that it’s invention could apply to other languages that had addressable *data* elements such as *Java* or *C++*. See col. 6, lines 10-11 of Chilimbi. Furthermore, as already discussed above, with respect to the Java environment example in cols. 10-11, Chilimbi discloses field reordering where a field is defined as a class instance *variable* which is a data element (rather than code or method) as is well known to those skilled in the art. Therefore, the office action’s assertion that Chilimbi reference to Java programs somehow teaches code caches and partitioning of instruction translations between hot and cold partitions of a code cache is clearly incorrect.

(C) In view of the above reasons, applicants respectfully submit that the independent claims 1, 13, and 23 and remaining claims in group 1, which depend therefrom, are patentable over the applied prior art.

Group 2

Dependent claims 10 and 20 are also believed to be patentable for at least the same reasons as discussed above with respect to Group 1 because they depend from independent claims 1 and 13, respectively, and are, therefore, patentable for at least the same reasons.

In addition, claims 10 and 20 recite determining whether a hot instruction translation in the hot partition of the code cache memory exceeds a second threshold value and if so, it expands the size of the hot partition by adding thereto an expansion area contiguous to the hot partition. *First*, as discussed with respect to Group 1, Chilimbi does not disclose anything

related to a code cache. *Second*, with respect to this feature, the final office action cites to col. 2, lines 37-39 and fig. 1, item 20 of Chilimbi which only discloses a generic computing system and the office action then alleges that such a conventional computing system “capable of creating and maintaining numerous memory locations.” See last paragraph of page 6 of the final office action dated November 17, 2004.


Of course, such a generic disclosure which is “capable of” having the claimed feature does not meet the PTO’s burden to prove anticipation. The Patent Office (PTO) has the burden of proving each of the claimed features is shown by the prior art. An allegation that claimed subject matter is obvious (as essentially alleged here although the office action uses anticipation) requires a positive, concrete teaching in the prior art, such as would lead a person skilled in the art to choose the claimed combination from among many that might be comprehended by broad prior art teachings. The PTO’s review court has made it very clear that silence in a reference is hardly a substitute for clear and concrete evidence from which a conclusion of obviousness (or anticipation) might justifiably flow. See, e.g., *Application of Burt*, 356 F.2d 115, 121 (CCPA 1966).

Accordingly, applicants submit that the features recited in claims 10 and 20, and claims 11 and 21 which are dependent therefrom, are not disclosed or suggested by the applied prior art. Accordingly, the claims of Group 2 are patentable for this additional reason.

9. Conclusion

In view of above, appellants respectfully solicit the Honorable Board of Patent Appeals and Interferences to reverse the rejection of the pending claims and pass this application on to allowance.

Respectfully submitted,



William T. Ellis
Reg. No. 26,874

Aaron C. Chatterjee
Reg. No. 41,398

April 18, 2005 (Monday)
Date

(202) 672-5300

10. Claims Appendix

1. (Once Amended) A method for operating a code cache in a dynamic instruction translator comprising the steps of:

storing a plurality of instruction translations in a cold partition in a cache memory;
determining whether an instruction translation that has been stored in the cold partition is hot; and

moving the instruction translation to a hot partition in the cache memory when an instruction translation has been determined to be hot.

2. (Once Amended) A method as defined in claim 1, wherein the step of determining whether an instruction translation is hot comprises:

maintaining a different associated counter for each of a plurality of instruction translations in the cold partition of the cache memory;

incrementing or decrementing the count in the associated counter each time its associated instruction translation is executed; and

concluding the determination that an instruction translation is hot if the count in the associated counter reaches a first threshold value.

3. (Unamended) A method as defined in claim 1, wherein said hot partition is contiguous and disjoint from said cold partition in said cache memory.

4. (Unamended) A method as defined in claim 2, wherein said maintaining an associated counter step comprises maintaining counters in a data structure external to said cache memory.

5. (Once Amended) A method as defined in claim 4, further comprising the step of at least temporarily delinking blocks of instruction translations stored in said cold partition so that control exits the cache memory in order to perform the incrementing or decrementing step.

6. (Once Amended) A method as defined in claim 2, wherein said maintaining within said cache memory an associated counter step comprises maintaining one of said associated counters for each entry point into a plurality of the instruction translations in said cold partition of the cache memory.

7. (Once Amended) A method as defined in claim 2, wherein said maintaining an associated counter step comprises logically embedding update code on an arc between two instruction translations.

8. (Unamended) A method as defined in claim 2, wherein said maintaining an associated counter step comprises maintaining one of said associated counters for each machine cache line in an associated microprocessor.

9. (Once Amended) A method as defined in claim 2, wherein said instruction translation moving step comprises sampling a plurality of said associated counters on an intermittent basis to determine if the count therein has reached said threshold value.

10. (Once Amended) A method as defined in claim 1, further comprising the steps of:

determining if a number of hot instruction translations in said hot partition of said cache memory exceeds a second threshold value; and

if said number of said hot instruction translations exceeds said second threshold value, then expanding the size of said hot partition in said cache memory by adding thereto an expansion area contiguous to said hot partition.

11. (Once Amended) A method as defined in claim 10, further comprising the step of

removing all cold instruction translations from said expansion area and storing said removed instruction translations in said cold partition.

12. (Once Amended) A method as defined in claim 2, wherein the maintaining an associated counter step comprises maintaining an associated counter for all instruction translations in the cold partition of the cache memory.

13. (Once Amended) A system for a code cache in a dynamic instruction translator comprising:

- a cache memory;
- a cold partition and a hot partition in said cache memory;
- logic for determining whether an instruction translation that has been stored in the cold partition is hot; and
- logic for moving the instruction translation to a hot partition in the cache memory when an instruction translation has been determined to be hot.

14. (Once Amended) A system as defined in claim 13, wherein the logic for determining whether an instruction translation is hot comprises:

- logic for associating a different counter for each of a plurality of instruction translations stored in the cold partition of the cache memory;
- logic for incrementing or decrementing the count in the associated counter each time its associated instruction translation is executed; and
- logic determining if the count in the associated counter reaches a first threshold value.

15. (Unamended) A system as defined in claim 13, wherein said hot partition is contiguous and disjoint from said cold partition in said cache memory.

16. (Unamended) A system as defined in claim 14, wherein said counters are maintained in a data structure external to said cache memory.

17. (Once Amended) A system as defined in claim 16, wherein said incrementing or decrementing logic further comprises logic for at least temporarily delinking blocks of instruction translations stored in said cold partition so that control exits the cache memory in order to perform the incrementing or decrementing of the count.

18. (Once Amended) A system as defined in claim 14, wherein said logic for associating counters comprises logic for maintaining one of said associated counters for each entry point into a plurality of the instruction translations in said cold partition of the cache memory.

19. (Once Amended) A system as defined in claim 14, wherein said logic for moving the instruction translation comprises logic for sampling a plurality of said associated counters on an intermittent basis to determine if the count therein has reached said threshold value.

20. (Once Amended) A system as defined in claim 13, further comprising:
logic for determining if a number of hot instruction translations in said hot partition of said cache memory exceeds a second threshold value; and
if said number of said hot instruction translations exceeds said second threshold value, logic for expanding the size of said hot partition in said cache memory by adding thereto an expansion area contiguous to said hot partition.

21. (Once Amended) A system as defined in claim 20, further comprising:
logic for removing all cold instruction translations from said expansion area and storing said removed instruction translations in said cold partition.

22. (Once Amended) A system as defined in claim 14, wherein the logic for associating a counter step comprises logic for maintaining an associated counter for all instruction translations in the cold partition of the cache memory.

23. (Twice Amended) A program product, comprising a computer usable medium having computer readable program code embodied therein that, when executed, directs a computer to manage a code cache memory in a dynamic instruction translator, the program code comprising:

code for storing a plurality of instruction translations in a cold partition in a cache memory;

code for determining whether an instruction translation that has been stored in the cold partition is hot; and

code for moving the instruction translation to a hot partition in the cache memory when an instruction translation has been determined to be hot.